## AMENDMENTS TO THE CLAIMS

Kindly replace the claims as follows.

1. (currently amended) The computer, comprising:

a multi-stage execution pipeline;

an instruction decoder designed:

to decode instructions of a complex instruction set for execution by the pipeline, the instruction set being architecturally exposed for execution by user-state programs stored in a main memory of the computer, the instruction set having variable-length instructions and many instructions having multiple side-effects and a potential to raise multiple exceptions,

for at least some instructions of the complex instruction set, to issue two or more instructions in a second internal form into the execution pipeline;

to generate information descriptive of instructions to be executed by the pipeline, and to store the information into non-pipelined processor registers of the computer;

to determine whether instructions will complete in the pipeline, and to abstain from writing descriptive information into the processor registers for instructions following an instruction determined not to complete;

the pipeline being designed to recognize an exception occurring in an instruction after an earlier ~~a first~~ side-effect of the instruction has been architecturally committed and before a later side of the instruction is architecturally committed, and thereupon, to architecturally expose in the processor registers information describing a processor state of the computer, including an intra-instruction program counter value, and to transfer execution to an exception handler; and

pipeline resumption circuitry effective after completion of the software exception handler to resume execution of the excepted program based on the information in the processor registers;

the processor registers of the computer being designed to architecturally expose sufficient information about the state of the excepted instruction that the transfer and resume are effected without saving processor state to the main memory.

2. (original) The method, comprising the steps of:

decoding and executing instructions of a complex instruction set of a computer, the instruction set having variable-length instructions and many instructions having multiple side-effects;

storing information describing the decoding of the complex instructions into architecturally-visible processor registers of the computer.

3. (currently amended) The method of claim 2, further comprising the step of:

recognizing an exception occurring in an instruction after an earlier ~~a first~~ side-effect of an instruction has been architecturally committed and before a later side of the instruction is architecturally committed;

transferring control to a software exception handler for the ~~first~~ exception;

resuming execution of the excepted instruction after completion of the exception handler, the architecturally-visible processor registers exposing sufficient information that the transfer and resumption are effected without saving the full state of the excepted instruction on a memory stack.

4. (original) The method of claim 2:

wherein the instructions are executed in a multi-stage execution pipeline, and the architecturally-visible processor registers are not pipelined;

and further comprising the steps of:

determining whether instructions will complete in the pipeline; and

abstaining from writing descriptive information into the architecturally-visible processor registers for instructions following an instruction determined not to complete.

5. (currently amended) The method of claim 2, further comprising the steps of:

on recognizing an exception occurring in an instruction after an earlier ~~a first~~ side-effect of an instruction has been architecturally committed and before a later side of the instruction is architecturally committed, architecturally exposing an instruction pointer, contents of a general register file, and contents of processor registers to a software exception handler, the processor registers and general purpose registers of the computer architecturally exposing sufficient processor state and providing sufficient working storage for execution of a software exception handler and resumption of the program, without recomputing the earlier ~~first~~ side-effect, and without storing processor state to the main memory.

6. (original) The method of claim 5, further comprising the step of:

abstaining the storing during execution of the software exception handler.

7. (original) The method of claim 2, wherein:

wherein the complex instructions are fetched in an external form from a memory;

and further comprising the steps of:

for at least some external-form instructions, decoding each instruction into two or more instructions into an internal form for execution in an execution pipeline;

capturing and architecturally exposing an intra-instruction program counter value into the architecturally-visible processor register when a complex instruction raises an exception at an intermediate point.

8. (original) The computer, comprising:

an instruction decoder and pipeline designed to decode and execute instructions of a complex instruction set having variable-length instructions and many instructions having multiple side-effects;

processor register control circuitry designed to store information describing the decoding of the complex instructions into architecturally-visible processor registers of the computer.

9. (original) The computer of claim 8, further comprising:

a collection of software exception handlers, the processor register circuitry designed to abstain from storing information into the processor registers during execution of at least some of the software exception handlers.

10. (original) The computer of claim 8, wherein the decoding information presents information about the instructions of the complex instruction set in a form uniform across most of the complex instruction set.

11. (original) The computer of claim 8, further comprising a mask register of bits, each bit corresponding to a class of instructions of the instruction set, a value of each bit designating whether to raise an exception on execution of an instruction of the corresponding class.

12. (original) The computer of claim 8, wherein the architecturally-visible processor registers are not architecturally-visible in the complex instruction set, but are architecturally-visible in an alternative instruction set of the computer, the alternative instruction set being architecturally available to user-state programs.

13. (original) The computer of claim 8, wherein the decoding information includes a designation of any prefix to the current instruction.

14. (original) The computer of claim 8, wherein the decoding information includes a designation of an operand effective address.

15. (original) The computer of claim 8, wherein the decoding information includes a sign-extended value of an immediate value.

16. (original) The computer of claim 8, wherein the decoding information includes a designation of a length of the currently-executing instruction.

17. (original) The computer of claim 8, wherein the decoding information includes a designation of a current instruction pointer and an instruction pointer to a next instruction.

18. (original) The computer of claim 8, wherein the decoding information includes an instruction pointer value and an indication of an intra-instruction fractional completion of the complex instructions.

19. (original) The computer of claim 8, wherein the decoding information includes an instruction pointer value and an indication of a protection mode of the computer.

20. (currently amended) The computer of claim 8, further comprising:

pipeline control circuitry designed to recognize an exception occurring in an instruction after an earlier ~~a first~~ side-effect of the instruction has been architecturally committed and before a later side of the instruction is architecturally committed, to transfer control to a software exception handler for the ~~first~~ exception, and to resume execution of the excepted instruction after completion of the exception handler, processor registers of the computer being designed to architecturally expose sufficient information about the state of the excepted instruction that the transfer and resume are effected without saving intermediate results of the excepted instruction on a memory stack.

21. (currently amended) The method, comprising the steps of:

executing a program in user state of a computer, the program coded in an instruction set having many instructions with multiple side-effects and the potential to raise multiple exceptions;

in response to recognizing an exception occurring in an instruction after an earlier ~~a first~~ side-effect of the instruction has been architecturally committed and before a later side-effect of the instruction is architecturally committed, transferring control to a software exception handler for the ~~first~~ exception, and resuming execution of the excepted instruction after completion of the exception handler, processor registers of the computer being designed to architecturally expose sufficient information about the intermediate state of the excepted instruction that the transfer and resume are effected without saving intermediate results of the excepted instruction on a memory stack.

22. (original) The method of claim 21, further comprising the steps of:

wherein the instructions are executed in a multi-stage execution pipeline, and the processor registers are not pipelined;

and further comprising the steps of:

at an early stage of the pipeline, determining whether instructions will complete in the pipeline; and

abstaining from writing descriptive information into the architecturally-visible processor registers for instructions following an instruction determined not to complete.

23. (currently amended) The method of claim 21, further comprising the steps of:

on recognizing the exception, architecturally exposing an instruction pointer, contents of a general register file, and contents of processor registers to a software exception handler, the processor registers and general purpose registers of the computer architecturally exposing sufficient processor state and providing sufficient working storage for execution of a

software exception handler and resumption of the program, without recomputing the earlier ~~first~~ side-effect, and without storing processor state to the main memory.

24. (currently amended) The method of claim 21, further comprising the steps of:

in response to recognizing an exception occurring in an instruction after the earlier ~~a~~ ~~first~~ side-effect of the instruction has been architecturally committed, architecturally exposing an intra-instruction program counter value in the processor registers.

25. (currently amended) The computer, comprising:

an instruction decoder for an instruction set exposed for execution by user-state programs, many individual instructions of the instruction set having multiple side-effects and a potential to raise multiple exceptions;

pipeline control circuitry designed to recognize an exception occurring in an instruction after an earlier ~~a first~~ side-effect of the instruction has been architecturally committed and before a later side-effect of the instruction is architecturally committed, to transfer control to a software exception handler for the earlier ~~first~~ exception, and to resume execution of the excepted instruction after completion of the exception handler, processor registers of the computer being designed to architecturally expose sufficient information about the state of the excepted instruction that the transfer and resume are effected without saving intermediate results of the excepted instruction on a memory stack.

26. (original) The computer of claim 25, further comprising:

processor register control circuitry designed to store information describing decoding of sequential instructions decoded by the decoder into the processor registers of the computer.

27. (original) The computer of claim 25, wherein the processor registers are not architecturally-visible in the multiple side-effect instruction set, but are architecturally-visible

in an alternative instruction set of the computer, the alternative instruction set being architecturally available to user-state programs.

28. (original) The computer of claim 25, wherein the pipeline control circuitry is further designed to abstain from the altering the contents of the processor registers during execution of the exception handler.

29. (original) The computer of claim 25, wherein the decoding information includes a designation of a base register and offset of an operand effective address.

30. (original) The computer of claim 25, wherein the decoding information includes a designation of a current instruction pointer and an instruction pointer to a next instruction.

31. (original) The computer of claim 25, wherein the decoding information includes an instruction pointer value and an indication of an intra-instruction fractional completion of the complex instructions.

32. (original) The computer of claim 31, wherein the an indication of an intra-instruction fractional completion includes a designation of a repeat prefix to the current instruction.

33. (original) The computer of claim 25, wherein the operation of the exception handler is controlled at least in part by the contents of the processor registers.

34. (original) The computer of claim 25, wherein intermediate results of the multiple side-effect instructions are stored in general purpose registers of the computer.

35. (original) The method, comprising the step of:

while decoding a sequence of computer instructions for execution in a multi-stage execution pipeline and before commencing substantial execution of each decoded instruction of the sequence, generating information descriptive of the instruction, and, depending on a determination of whether the instruction will complete in the pipeline, storing or abstaining from storing the generated information into a non-pipelined register of the computer.

36. (currently amended) The method of claim 35:

on recognizing an exception occurring in an instruction after an earlier ~~a first~~ side-effect of the instruction has been architecturally committed and before a later side of the instruction is architecturally committed, architecturally exposing an instruction pointer, contents of a general register file, and contents of processor registers to a software exception handler, the processor registers and general purpose registers of the computer architecturally exposing sufficient processor state and providing sufficient working storage for execution of a software exception handler and resumption of the program, without recomputing the earlier ~~first~~ side-effect, and without storing processor state to the main memory.

37. (original) The method of claim 35:

for at least some of the decoded instructions, issuing two or more instructions in a second instruction set into the pipeline;

for at least some of the decoded instructions, capturing and architecturally exposing an intra-instruction program counter value when the decoded instruction raises an exception at an intermediate point.

38. (original) The computer, comprising:

a multi-stage execution pipeline;

an instruction decoder designed to generate information descriptive of instructions to be executed by the pipeline, and to store the information into a non-pipelined register of the computer;

the instruction decoder being designed to determine whether instructions will complete in the pipeline, and to abstain from writing descriptive information into the register for instructions following an instruction determined not to complete.

39. (original) The computer of claim 38, wherein the instruction decoder and pipeline are designed to decode and execute instructions of a complex instruction set having variable-length instructions and many instructions having multiple side-effects.

40. (original) The computer of claim 39, wherein the descriptive information describes the decoding of the complex instructions.

41. (currently amended) The computer of claim 38, further comprising:
pipeline control circuitry designed to recognize an exception occurring in an instruction after an earlier ~~a first~~ side-effect of the instruction has been architecturally committed and before a later side of the instruction is architecturally committed, to transfer control to a software exception handler for the ~~first~~ exception, and to resume execution of the excepted instruction after completion of the exception handler, processor registers of the computer being designed to capture sufficient information about the state of the excepted instruction that the transfer and resume are effected without saving intermediate results of the excepted instruction on a memory stack.

42. (original) The computer of claim 38, wherein the execution pipeline and instruction decoder are designed to retire instructions individually and independently, with at most a few interactions between instructions to affect retirement.

43. (original) The computer of claim 38, wherein:

intermediate results of instructions decoded by the instruction decoder are stored in registers of a general register file that are not architecturally addressable in the instructions decoded by the instruction decoder.

44. (original) The computer of claim 38, wherein the decoding information includes an instruction pointer value and an indication of an intra-instruction fractional completion of the complex instructions.

45. (original) The computer of claim 38, further comprising a mask register of bits, each bit corresponding to a class of instructions of the instruction set, a value of each bit designating whether to raise an exception on execution of an instruction of the corresponding class.

46. (original) The method, comprising the steps of:

while executing a program coded in an instruction set exposed for execution by programs stored in a main memory of the computer, recognizing an exception occurring in a program, and in response, architecturally exposing in processor registers of the computer information describing a processor state of the computer and transferring execution to an exception handler;

after completion of the software exception handler, resuming execution of the excepted program based on the information in the processor registers;

the processor registers and general purpose registers of the computer architecturally exposing sufficient processor state and providing sufficient working storage for execution of

the exception handler and resumption of the program, without storing processor state to the main memory.

47. (original) The method of claim 46, wherein the exception occurs at an intermediate point of an instruction, and the processor registers architecturally expose an intra-instruction program counter value.

48. (original) The method of claim 46, wherein instructions of the exposed instruction set are managed by a memory management unit between a main memory of the computer and one or more cache levels.

49. (original) The computer, comprising:

an instruction decoder for an instruction set exposed for execution by programs stored in a main memory of the computer;

pipeline exception circuitry, effective on recognizing an exception occurring in a stored program, to architecturally expose in processor registers of the computer information describing a processor state of the computer, and to transfer execution to an exception handler; and

pipeline resumption circuitry effective after completion of the software exception handler to resume execution of the excepted program based on the information in the processor registers;

the processor registers and general purpose registers of the computer architecturally exposing sufficient processor state and providing sufficient working storage for execution of the exception handler and resumption of the program, without storing processor state to the main memory.

50. (original) The computer of claim 49, wherein:

the instruction set is a complex instruction set having variable-length instructions and many instructions having multiple side-effects and a potential to raise multiple exceptions.

51. (original) The computer of claim 50, wherein:

the processor registers are designed to describe the decoding of the complex instructions.

52. (original) The computer of claim 49, wherein:

the exception occurs at an intermediate point in the execution of an instruction; and

the processor registers of the computer are designed to capture sufficient information about the state of an the excepted instruction that the transfer and resume are effected without saving intermediate results of the excepted instruction on a memory stack.

53. (original) The computer of claim 49, wherein:

an execution pipeline of the computer is arranged in a plurality of stages;

the processor registers are not pipelined; and

the instruction decoder is designed to determine whether instructions will complete in the pipeline, and to abstain from writing descriptive information into the processor registers for instructions following an instruction determined not to complete.

54. (original) The computer of claim 49, wherein the execution pipeline and instruction decoder are designed to retire instructions individually and independently, with at most a few interactions between instructions to affect retirement.

55. (original) The computer of claim 49, wherein:

intermediate results of instructions decoded by the instruction decoder are stored in registers of a general register file that are not architecturally addressable in the instruction set decoded by the instruction decoder.

56. (original) The computer of claim 49, wherein the information stored in the processor registers includes a designation of a current instruction pointer and an instruction pointer to a next instruction.

57. (original) The computer of claim 49, further comprising a mask register of bits, each bit corresponding to a class of instructions of the instruction set, a value of each bit designating whether to raise an exception on execution of an instruction of the corresponding class.

58. (original) The computer of claim 49, being designed to preserve values of the processor registers during execution of the software exception handler.

59. (original) The method, comprising the steps of:

fetching instructions in a first external instruction set from a memory, and, for at least some instructions of the first instruction set, issuing two or more instructions in a second form into an execution pipeline;

architecturally exposing an intra-instruction program counter value when an instruction of the first instruction set raises an exception at an intermediate point.

60. (original) The method of claim 59, wherein a register exposing the intra-instruction program counter is not architecturally visible in the first instruction set, but is architecturally-visible in the second instruction set of the computer, the second instruction set being architecturally available to user-state programs.

61. (original) The computer, comprising:

an instruction decoder and execution pipeline, the decoder designed to fetch instructions in a first instruction set from a memory, and, for at least some instructions of the first instruction set, to issue two or more instructions in a second internal form into the execution pipeline;

a register and control logic for that register that capture and architecturally expose an intra-instruction program counter value when an instruction of the first instruction set raises an exception at an intermediate point.

62. (original) The computer of claim 61 wherein the first instruction set is a complex instruction set having many instructions having multiple side-effects and the potential to raise multiple exceptions.

63. (original) The computer of claim 62, further comprising:

processor register control circuitry designed to store information describing the decoding of the complex instructions into architecturally-visible processor registers of the computer.

64. (currently amended) The computer of claim 62, further comprising:

pipeline control circuitry designed to recognize an exception occurring in a complex instruction after an earlier ~~a first~~ side-effect of the complex instruction has been architecturally committed and before a later side of the instruction is architecturally committed, to transfer control to a software exception handler for the ~~first~~ exception, and to resume execution of the excepted complex instruction after completion of the exception handler, processor registers of the computer being designed to architecturally expose sufficient information about the state of the excepted complex instruction that the transfer and resume are effected without saving intermediate results of the excepted instruction on a memory stack.

65. (currently amended) The computer of claim 62, further comprising:

pipeline exception circuitry, designed to recognize an exception occurring in an instruction after an earlier ~~a first~~ side-effect of an instruction has been architecturally committed and before a later side-effect of the instruction is architecturally committed, and in response, to architecturally expose an instruction pointer, contents of a general register file, and contents of processor registers to a software exception handler, the processor registers and general purpose registers of the computer architecturally exposing sufficient processor state and providing sufficient working storage for execution of a software exception handler and resumption of the program, without recomputing the earlier ~~first~~ side-effect, and without storing processor state to the main memory.

66. (original) The computer of claim 61 wherein:

the execution pipeline is a multi-stage execution pipeline;

the instruction decoder is designed to generate information descriptive of the instructions of the first instruction set to be executed by the pipeline, and to store the information into a non-pipelined register of the computer; and

the instruction decoder is designed to determine whether instructions of the first instruction set will complete in the pipeline, and to abstain from writing descriptive information into the register for instructions following an instruction determined not to complete.

67. (original) The computer of claim 61, further comprising:

a software exception handler coded to determine a location of an operand of the instruction based on the intra-instruction program counter value.

68. (currently amended) The computer of claim 61, further comprising:

pipeline resumption circuitry is designed to restart the excepted instruction recognize an exception occurring in a CISC instruction after an earlier ~~a first~~ side-effect of the CISC instruction has been architecturally committed and before a later side-effect of the instruction is architecturally committed, and thereupon, to architecturally expose an instruction pointer,

contents of a general register file, and contents of processor registers to a software exception handler, the processor registers and general purpose registers of the computer architecturally exposing sufficient processor state and providing sufficient working storage for execution of the exception handler and resumption of the program, without recomputing the earlier ~~first~~ side-effect, without storing processor state to the main memory.

69. (original) The computer of claim 61, wherein the intra-instruction program counter value is a serial count of instructions issued by the instruction decoder in response to decoding an instruction of the first instruction set.

70. (original) The computer of claim 61, wherein the intra-instruction program counter value has a reserved value to indicate that the instruction decoder is currently in a mode to fetch instructions in the second form from a memory of the computer.

71. (original) The computer of claim 61, further comprising:

processor registers and control circuitry therefor designed to capture an intermediate state of an excepted instruction in sufficient detail for restart of the excepted instruction without rerunning the portion of the instruction that completed before the exception, and without the intermediate state being spilled to memory.